A Library of OpenGL-Based Mathematical Image Filters

Martin von Gagern, Christian Mercat

A Library of OpenGL-based Mathematical Image Filters

Martin von Gagern[1] and Christian Mercat[2]

[1] Zentrum Mathematik (M10), TU München, 85748 Garching, Germany
[2] S2HEP, UCB Lyon 1, 69622 Villeurbanne cedex, France

Abstract. There are a lot of transformations that can turn one raster image into a derived one in a mathematically interesting way. This article describes a collection of such filters, implemented in OpenGL in order to use the high degree of parallelism modern GPUs provide, thereby providing performance required to process e.g. live camera images in real-time. The filters contained in this library include wallpaper groups, conformal maps described by meromorphic functions, as well as hyperbolic symmetry groups. Using examples of increasing complexity, several key implementation techniques are explained, including texture wrap configurations, user-configurable control points, and custom fragment shader programs. This work might exhibit aesthetic aspects of mathematics to the masses and provide useful building blocks for scientists as well as artists.

Key words: Transformation, tiling, wallpaper group, conformal map, meromorphic function, hyperbolic geometry, GPU, parallelism, webcam

1 Introduction

1.1 Definition

In this article, an *image filter* is a piece of program code which can transform an input raster image to generate an output raster image.

Most filters have a set of parameters tuning their behavior. Besides common numeric parameters, control points are of special interest. A control point is a point that can be moved around in the input image and that will influence the resulting output image. Parameters can be interactive, so that modifying one parameter might implicitly adjust other parameters as well. Some filters might allow mathematical formulas as parameter input.

Our image filters are implemented in OpenGL so that most of the actual image transformation can be computed on the GPU. With this approach we expect to be able to provide sizable result images based for example on a live video stream from a webcam, while maintaining a decent frame rate.

1.2 Motivation and Prior Art

The basic ideas for this collection of image filters come from a number of previous projects. There is *morenaments euc*, an application written by Martin von Gagern which allows a user to draw in any of the 17 wallpaper groups of the Euclidean plane. From that project derived *morenaments hyp*, a similar application for hyperbolic ornaments, for which some essential design principles were demonstrated at the ICMS 2006.

The combination of these two areas led to the investigation of possible conformal transformations of Euclidean ornaments into hyperbolic ones, in close cooperation with Jürgen Richter-Gebert. In the course of this project the use of OpenGL fragment shader programs for rendering of hyperbolic ornaments was developed as well.

Christian Mercat is author of *Conformal Webcam*[1], a Java application applying a freely chosen meromorphic function to a live image from a webcam.

Each author's project could benefit a lot from the ideas of the other. Evaluating the complex function for the *Conformal Webcam* in a fragment shader program would greatly increase performance and therefore mitigate the problem of slow frame rates. On the other hand, using a live image as input for a hyperbolic ornament would give another stunning visual effect, impressively exhibiting the performance of this GPU-based approach. This gave rise to the ideas outlined below.

1.3 Composition of the Library

We intend to build a library of image filters which are of pedagogical or artistic interest. The idea is to visualize an abstract mathematical notion as the deformation of a picture.

In the long run the library will likely cover topics from complex analysis, crystallography, group theory, hyperbolic geometry, differential equations, differential geometry, and surfaces.

Currently, the real-time computability and the use of OpenGL are central criteria for the implementation as part of this library. However, in the long run it might make sense to extend the library beyond what is possible under these restrictions.

In the beginning the collection of filters will consist of two major groups: *symmetric ornaments* and *conformal maps*. Symmetric ornaments include the well known 17 Euclidean wallpaper groups as well as the infinite number of hyperbolic counterparts.

The filters for conformal maps describe transformations of the plane onto itself via an arbitrary meromorphic function. This allows visualization of a large class of interesting non-linear transformations.

Other groups of filters are likely to be added in the future.

1.4 A Word on Symmetry

The image filters described below will conceptually map a single rectangular image to the whole (Euclidean or hyperbolic) plane, usually using some form of repetition. At least in the case of the Euclidean plane, the resulting image is a portion of the infinite result. This repetitiveness is not a strong requirement for future additions to the library, though.

When describing the form such a repetition takes, it is useful to name the underlying symmetry group. In most cases, this will be one of the 17 wallpaper groups of the Euclidean plane. This article will usually state both the crystallographic group name (starting with a letter p or c) as well as the corresponding *orbifold symbol*.[2] The latter provide a concept which easily extends to the hyperbolic plane as well.

1.5 Short Introduction to OpenGL

OpenGL is a complex and powerful graphics API. This section introduces only those aspects relevant to explanations below. While the primary application of OpenGL is the preferably photorealistic rendering of three-dimensional scenes, it can be used to render two-dimensional abstract images as well, simply by placing all objects in a plane, using an orthogonal projection and disabling all lighting effects.

An OpenGL application describes a scene as a set of geometric primitives, e.g. triangles. Attributes can be associated with each vertex of these primitives. The attribute most important here is the so-called texture coordinate. After loading an image as a texture, these texture coordinates can be used to fill the interior of the primitive by projective interpolation of the corresponding part of the texture image.

The texture interpolation process described above is the common case, called the *fixed functionality pipeline* and hard-wired into older graphics cards. More recent hardware supports the application to replace this fixed functionality with custom code, called a *fragment shader program*. This allows for very powerful calculations. The important fact is that these calculations are executed on the GPU for several pixels in parallel. Due to the large number of shading units on the GPU, this tends to be a lot faster than the same calculations would be if calculated on a single CPU core. Exploiting this parallelism for mathematical visualization is one of the aims of the project described in this article.

Each instance of a fragment shader program is responsible for the color of a single pixel (except when supersampling). As a consequence of the parallelism, there can be no communication between these programs, so the program should be structured in such a way that it can handle calculations for every pixel independently. Shader programs for OpenGL are written in the OpenGL Shading Language, which looks a lot like C, but provides built-in support for vectors, matrices and other things useful for geometric calculations. The source code of this program is compiled by the graphics card driver software.

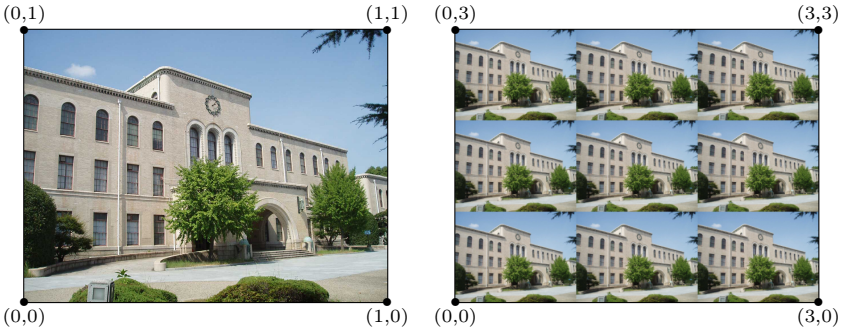
2 Explanation by Examples

Some things are best explained by example. So this section will describe a number of image filters in some detail, in order to demonstrate implementation techniques employed by other filters as well. The examples start simple, and their descriptions build on those of preceding examples.

2.1 Tiled Rectangles and Wraparound Parameters

Probably one of the simplest transformations is *simple tiling*. Copies of a rectangular input image are placed adjacently to cover the whole plane. The underlying symmetry group consists only of translations. It is the wallpaper group p1, also denoted with the orbifold symbol ∘.

Implementing this kind of transformation is very simple indeed. The input image is loaded into the OpenGL context as a texture. An orthogonal projection is chosen for the result image. The scene, as described in OpenGL, can consist of a single quad, spanning the area of the result image. Suitable texture coordinates for the corners of the quad will determine the number of repeats of the texture within the result image. The texture has to be configured with `GL_REPEAT` wrap parameters for both directions.



(a) The input texture (b) A simple tiling

Fig. 1: A simple tiling with texture coordinates

As a variation of this approach, one can replicate the input image using reflections instead of translations along one or both its axes. Translations in one direction and reflections in the other give the wallpaper group pm (orbifold symbol ∗∗), while reflections along all the edges of the input image yield the wallpaper group pmm (orbifold symbol ∗2222). Both of these are implemented using the above setup with `GL_MIRRORED_REPEAT` instead of `GL_REPEAT` for one or both directions.

(a) Wallpaper group pm

(b) Wallpaper group pmm

Fig. 2: Using different reflections

The techniques presented up to here are far from new; many computer games employ the wraparound parameters to render all kinds of background textures, including wallpapers.

2.2 Wallpaper Groups and Fixed Functionality Pipeline

To make things more interesting, let's look at *kaleidoscopes* next. A kaleidoscope consists of a set of mirrors arranged in such a way that the generated reflections line up and form a periodic pattern. Replacing the physical mirrors with mirror reflections, the whole setup can be flattened into the Euclidean plane.

There are four possible kaleidoscope groups. Three of them use three mirrors, while the last one requires four mirrors.

Interior angles	$(\frac{\pi}{3}, \frac{\pi}{3}, \frac{\pi}{3})$	$(\frac{\pi}{4}, \frac{\pi}{2}, \frac{\pi}{2})$	$(\frac{\pi}{6}, \frac{\pi}{3}, \frac{\pi}{2})$	$(\frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2})$
Symmetry group	p3m1	p4m	p6m	pmm
Orbifold symbol	$*333$	$*442$	$*632$	$*2222$

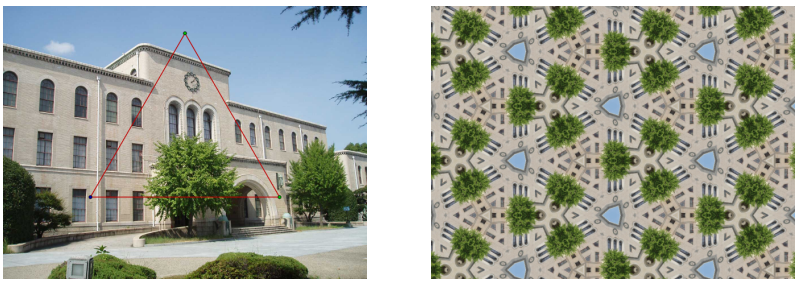
Table 1: Kaleidoscope groups

While the group pmm might be applied to the whole rectangular input image, the kaleidoscopes using three mirrors cut out a triangular part of any image, to be used as the fundamental domain of the resulting symmetric pattern. This is where parameters come into play: we want to give users the freedom to choose the section of the input image on which they want to operate. This is best achieved by three control points, corresponding to the three corners of the triangle.

Moving one control point should move the triangle as a whole, while changes to the other two control points can be used to control size and orientation of the selected triangle. The shape is determined by the symmetry group and will remain fixed. Obviously, adjusting one of these points will adjust the others as

well, which is the reason why control points are not independent input parameters, but instead modification of one will influence the others as well. The user interface has to take this kind of interaction into account.

In the OpenGL scene, this kind of pattern can no longer be expressed using a single quad and suitable wrap parameters for the input texture. Instead, every copy of the fundamental domain has to be expressed separately, using one triangle or quad each. The texture coordinates associated with the vertices align it with the portion of the input texture selected using the control points.



(a) Input with control points (b) Output

Fig. 3: The wallpaper group p3m1

Moving from kaleidoscopes to general wallpaper groups, new challenges arise. We'll demonstrate them using the wallpaper group p3 (orbifold symbol 333) as an example. That's the symmetry group obtained by taking only the orientation-preserving elements of p3m1.

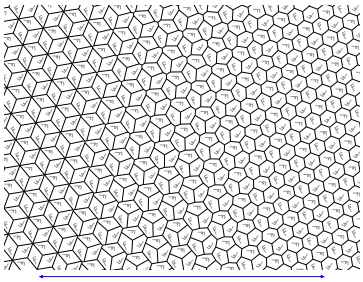


Fig. 4: The spectrum of fundamental domains for p3

The fundamental domain of p3 has twice the size of that of p3m1. But where the shape of the fundamental domain of p3m1 had to be an isosceles triangle, bounded by the lines of reflection, no such restriction applies to p3. Even if we

restrict the fundamental domain to be a convex polygon, there is still a whole spectrum of possible shapes, illustrated in Figure 4. Of these, the rhombus and the regular hexagon are probably the most regular and thus the most aesthetic ones, but certainly not the only possibilities.

This calls for yet another set of three control points, to control the shape of the fundamental domain. Moving any one of these three points will move the others as well.

While most applications probably never use it, it is possible to draw arbitrary convex polygons in OpenGL. Therefore the scene description can be implemented in a straight-forward way, without having to cut the polygons into triangles first.

Similar techniques can be used to implement all 17 wallpaper groups.

They all rely only on the so called fixed functionality pipeline: textures and geometric primitives only, without any custom shader code. This is because the map from destination positions to source positions is piecewise projective, in this case even piecewise affine.

2.3 Conformal Maps and Programmable Vertex Shaders

Everybody is used to visualizing a function from the plane to the real numbers, like precipitation maps: simply color the target space $\mathbb{R}$ with colors and plot each point (x, y) of the domain space $\mathbb{R}^2$ by the color $f(x, y)$.

A generic *conformal filter* operates on $\mathbb{C}$ instead of $\mathbb{R}^2$. It simply paints a point z of the domain space (i.e. output image) with the color of the point $f(z)$ in the target space (i.e. input image) where f is some meromorphic function such as tan, exp, or log. Most keys on a calculator are seen as real functions but have an analytic continuation as a meromorphic function.

The complex differentiability is visualized by the fact that the picture in the domain space, away from singularities, is to the first order around z a simple similitude of parameter $1/f'(z)$, since locally the function behave as $f(z+z_0) = f(z_0) + z \cdot f'(z_0) + o(|z|)$. In particular, the zeros of the derivative are very easy to spot as the similitude ratio tends to infinity. There, the function is no longer conformal, it behaves locally as a monomial, $f(z+z_0)-f(z_0) = \frac{f^{(k)}(z_0)}{k!}\, z^k + o(z^k)$ and the angles through z_0 are divided by k, replicating the features k times. Other striking points are the logarithmic singularities. Christian Mercat used his Conformal Webcam during a master course on complex analysis.

As these meromorphic functions cannot reasonably be expressed as piecewise projective maps, they have to be implemented using a different technique.

In order to get a whole plane from the single rectangular input, we can again wrap that input at its edges, with or without reflections, just as we did for the most simple tilings. Another thing we can take from these simple tilings is using a single quad spanning the output image as the whole scene description.

In order to map the texture onto that quad according to the exponential function, we have to replace the fixed functionality affine mapping with our own generic fragment shader. This fragment shader calculates suitable texture coordinates for every pixel of the result image by evaluation of the specified complex function. Then it determines the fragment color through a texture lookup.

(a) $f(z) = z^3 + c$ (b) $f(z) = \frac{\log z}{2\pi} \cdot (\frac{3}{4} + 3i)$

Fig. 5: Some conformal maps

Control points in the user interface could be passed as complex-valued constants into the formula, e.g. to configure specific features of the transformation, like the position of a singularity.

2.4 Basic Hyperbolic Tilings

A similar approach could be used for *tilings in the hyperbolic plane.*

In our opinion, the most aesthetic model of the hyperbolic plane is the Poincaré disk model. It represents the whole hyperbolic plane as the interior of the unit circle. Hyperbolic lines are modeled as circle arcs perpendicular to the unit circle, including lines through the origin as the special case of circles with infinite radius. The hyperbolic angle measure corresponds to the Euclidean angle between tangents. Due to this fact the model is conformal. The measure of length appears distorted, so that lines of equal hyperbolic length appear smaller the closer they are to the rim of the unit circle.

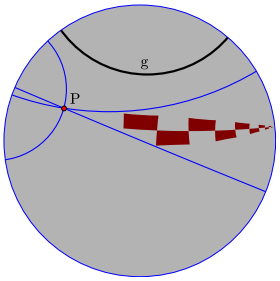


Fig. 6: The Poincaré model of the hyperbolic plane, illustrating multiple lines through P not intersecting g as well as a rule with steps of equal hyperbolic length

Orientation-preserving isometric transformations of the hyperbolic plane can be expressed as projective transformations of the complex line. Orientation-reversing transformations additionally require complex conjugation. To be more precise, we describe hyperbolic transformations using this formula:

$$\begin{pmatrix} x+yi \\ z+wi \end{pmatrix} \mapsto \begin{pmatrix} c-di & a+bi \\ a-bi & c+di \end{pmatrix} \cdot \begin{pmatrix} x+pyi \\ z+pwi \end{pmatrix} \qquad \begin{matrix} x,y,z,w \in \mathbb{R} \\ a,b,c,d \in \mathbb{R} \\ p \in \{-1,1\} \end{matrix} \tag{1}$$

where a, b, c, d, p characterize the transformation and x, y, z, w represent coordinates of a point in $\mathbb{CP}^1$. Scaling transformation matrices so that their determinants equal 1 (i.e. choosing a representative from $\mathrm{SL}(2, \mathbb{C})$) will help to avoid overflow when combining several transformations.

In order to render a hyperbolic tiling, we choose one fundamental domain at the center of the unit disk as the one providing the image data. The custom shader code will repeatedly apply generators of the group until the resulting position falls within that fundamental domain. Then a texture containing at least that fundamental domain is used to provide the actual color information.

In a crude approach, the fundamental domain can be simply cut from the input image. That means that a portion of the input image, delineated by circle arcs, is simply interpreted as a portion of the Poincaré disk. Figure 8(a) illustrates this method. However, while being simple and fast, this approach is somewhat clumsy from the mathematical point of view. Hyperbolic isometries which change the placement of the fundamental domain within the unit disc also affect the shape of the portion cut out from the input image, and the interpretation of that portion with respect to the hyperbolic distance measure.

2.5 Conformal Hyperbolization and Chained Transformations

More elegant would be an explicit and mathematically sound transformation step between the Euclidean input image and the hyperbolic fundamental domain. Conformality turns out to be a suitable requirement for such a transformation. On the one hand, as the angle measure is the same for the Euclidean plane and the Poincaré model of the hyperbolic plane, conformality is a concept that easily bridges the gap between both worlds. On the other hand, angles between objects have a great impact on how we perceive an image composition. So preserving angles will also preserve a lot of how an image actually looks to the human eye.

There is a way to conformally turn the fundamental domain of an Euclidean ornament into a fundamental domain for a related hyperbolic ornament[3]. The orbifold of the resulting hyperbolic ornament has the same topology as that of the Euclidean ornament, but different combinatorics. In other words, it changes (some of) the numbers in an orbifold symbol, while leaving all other parts of the orbifold symbol unchanged. In general, as the sum of interior angles in hyperbolic geometry is always smaller than in Euclidean geometry, the angles have to decrease and the numbers indicating the order of rotational centers therefore have to increase. However, if some of the numbers increase enough, it is possible for a single number to decrease as well.

The method used for the actual transformation of the fundamental domain comes from the field of discrete differential geometry. The central concept is called discrete conformal equality of triangle meshes[4], which I'll briefly describe here. Input consists of a triangle mesh, including full combinatoric information as well as the lengths of all edges, and a target angle sum for each vertex. There exists an algorithm to calculate a scale factor for each vertex, such that scaling all edges by the factors associated with both their endpoints will result in a new mesh, which will have the desired angle sums.

In particular, if the original triangulation corresponds to a topological disc with designated corners, and if furthermore inner vertices are assigned a target angle sum of 2π and edge vertices an angle sum of π, then corner angles can be chosen at will, and the result will be a flat polygon with straight edges and the desired angles at the corners.

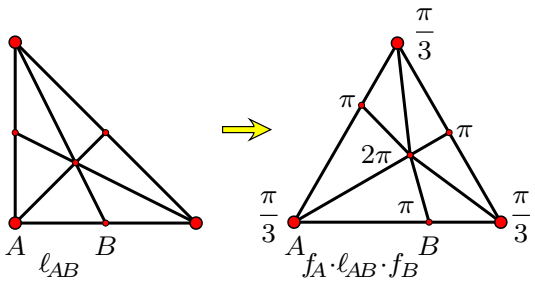


Fig. 7: Small example of a discretely conformal map

This association of scale factors with vertices of a mesh is in a way a discretization of the concept of continuous conformality. If the triangulation is sufficiently fine, the map between both meshes will be close to what a continuous conformal map would produce.

I will not go into the details of the algorithm. Suffice it to say that it is based on convex optimization. We are using an implementation of that algorithm called Confoo, written by Martin von Gagern.

After the triangle mesh has been transformed, the interior of the triangles has to be mapped as well. It turns out that superior to a simple affine interpolation is a specific projective interpolation, chosen such that it maps not only corners onto their images, but also the circumcircle of the source triangle to that of the target triangle. This kind of interpolation will be continuous along the edges of the triangles if and only if the two meshes are discretely conformally equivalent. The corresponding projective transformation can be easily computed from the scale factors associated with the vertices, and can be implemented in the projective geometry model provided by OpenGL.

Putting it all together, this gives the following processing chain: First the Euclidean input image is transformed into a hyperbolic image using discretely conformal triangle meshes. This can be achieved using the fixed functionality

pipeline, with the input image as a texture, the target mesh expressed using triangle primitives, and the texture coordinates chosen to express the required projective interpolation. The resulting image is stored in an off-screen buffer and then used in a second step as the texture for the calculation of a hyperbolic tiling using a fragment shader program.

(a) Fundamental domain simply cut from input image

(b) Conformal distortion of whole input image

Fig. 8: Hyperbolic patterns with orbifold symbol ∗3333

3 Applications

3.1 Edutainment

One major application of this library of filters is demonstrating mathematical concepts to the masses in a pleasing and fascinating way. It is fairly easy to combine a projection with a live camera feed, and as moving images tend to attract attention, and the interaction possible via the live image invites experimentation, it should be possible to rouse people's interest in the concepts behind these images. Seeing as many people still associate mathematics with dry formulas, it is important to show that the subject can be aesthetically beautiful as well.

3.2 Education

The pedagogical interest of such a setup, besides its appeal to students who see their faces distorted, is to be able to change on the spot the image in order to interactively show specific features of the transformation by simply pointing with the finger. The functional relation between the actual image and its deformation is rendered much more lively and concrete by the webcam animation, where a still image doesn't provide just enough information to make the connection.

In the classroom, some students will be named by their position on the plane for example, this one is called zero, this other one $+i$, or -1 and so on, they can hold signs to show to the camera. It helps making the learning process more tangible.

3.3 Arts

While a simple setup as described above might be suitable for scientific institutions, exhibitions and schools, artists can probably make a lot more of it by including these filters or their output into their own installations or other artworks. The authors would appreciate notice of such applications.

Some of the filters discussed above are already implemented as Pure-Data GEM plugins, but without using OpenGL and the benefits this can bring. Pure-Data is a programming environment used in particular by artists and performers to produce interactive sounds and images. It might be worthwhile to make the OpenGL versions of these filters available to users of Pure-Data in order to reach a wider audience.

For live performances, it would be nice if the artist could control the transformations, in particular the position of control points, using features detected in the input image. That way, an item moved around by the artist could actually influence the transformation, adding even more dynamic to the performance.

3.4 Derived Scientific Applications

Scientists might build on this library for their own applications. There are many example applications out there doing complex 3D scenes in OpenGL, but rather few examples of how to do elaborate 2D scientific computations in this way. This set of filters and the corresponding front end application provide such an example, and may serve as the basis for their own implementations, so they won't have to start from scratch.

All of these applications are freely available, as the library will be published under the GNU General Public License. Other licenses might be available from the authors upon request.

Acknowledgments. The authors would like to thank Jürgen Richter-Gebert for his contributions to the hyperbolization of ornaments, as well as for his valuable comments on a draft of this article. Thanks also to the user Hasec of Wikimedia Commons, who took the photo used as the example input throughout this paper.

References

1. Mercat, C.: Conformal webcam, Images des Math, CNRS, March 2009 `http://images.math.cnrs.fr/Applications-conformes.html`
2. Conway, J.H., Burgiel, H., Goodman-Strauss, C.: The Symmetries of Things. A K Peters, Wellesley (2008)
3. von Gagern, M., Richter-Gebert, J.: Hyperbolization of Euclidean Ornaments. Electronic Journal of Combinatorics 16(2), R12 (2009)
4. Springborn, B., Schröder, P., Pinkall, U.: Conformal equivalence of triangle meshes. ACM SIGGRAPH (2008)